Appendix B

3. CBR Data Structures

3.1. CBR Data Types

```
#define  CBR_VPN_DATA_TYPE        0
#define  CBR_VR_DATA_TYPE         1
#define  CBR_GROUP_DATA_TYPE      2
#define  CBR_OID_DATA_TYPE        3
```

3.2. CBR Message Tags

```
/* Control Blade Redundancy message tags */
#define  CBR_REPLY_TAG            0x6000
#define  CBR_MARK_TAG             0x6001
#define  CBR_DUMP_TAG             0x6003
#define  CBR_FINISH_TAG           0x6005
#define  CBR_ADD_TAG              0x6007
#define  CBR_DELETE_TAG           0x6009
#define  CBR_SB_DUMP_REQ_TAG      0x6011
#define  CBR_SEQ_TAG              0x6013
```

3.3. CBR Messages Format

```
typedef
struct   cbr_msg_s
{
   int               type;       /* Identifies Data type */
   int               seq;        /* Identifies sequence number */
   int               status;     /* return status */
}  cbr_msg_t;
```

3.4. CBR Node Structure

```
typedef
struct cbr_node_s
{
   TAG_DECL;                     /* run time type identification */
   dlcl_list_t       list;       /* List of all potential masters in the
system */
   address_space_t      addr;       /* Address space of peer */
#define CBR_READY                0
#define CBR_START_DUMP           1
#define CBR_DUMP_IN_PROGRESS     2
#define CBR_FINISH_DUMP          3
#define  CBR_UPDATE_ADD          4
#define  CBR_UPDATE_DELETE       5
   int               state;         /* State of the peer */
}  cbr_node_t;
```

[illegible]

0095348 3.091300

4. CBR API

4.1. CBR Init

```
void cbr_init ();
```

This function

1. Initializes list of the CBR nodes, which will communicate in the CBR process and adds itself to the list.
2. Registers callout function to the DML to be called in case message for CB Channel is received.
3. Initializes CBR message sequence numbers.

This function has to be called in every CBR processing node.

4.2. CBR Peer Up/Down

```
void cbr_peer_up (IN address_space_t addr);
```

1. Runs only on Master Control Blade
2. Does not run if peer up notification came with local address id (itself)
3. Adds peer to the CBR Node list
4. Starts dump database for peer

```
void cbr_peer_down (IN  address_space_t addr);
```

1. Runs only on Master Control Blade
2. Does not run if peer down notification came with the local address id (itself)
3. Removes peer from the CBR Node list

4.3. Update

Actions: CBR_ADD_ACTION and CBR_DELETE_ACTION

```
void cbr_master_update_oid (
    IN oid_link_t           *oid,
    IN int                  action);
```

Packages provided OID and sends packet to every Standby CB

```
void cbr_master_update_group (
    IN omorig_group_t       *grp,
    IN int                  action);
```

Packages provided GROUP and sends packet to every Standby CB

```
void cbr_master_update_vr (
    IN vr_descriptor_t       *vrdp,
    IN int                  action);
```

Packages provided VR and sends packet to every Standby CB

00671503.083402

```
void cbr_master_update_vpn (
   IN vpn_descriptor_t  *vpndp,
   IN int               action);
```
Packages provided VPN and sends packet to every Standby CB

These functions have to be called only on Master CB by OMORIG.

4.4. OM API used by Standby nodes

Standby nodes after receiving messages with the specified action for OM Global Database change make changes using the following set of API provided by the OM. The detailed explanation of API provided to the application is given in [5]. The detailed explanation of API, which is common to OM and CBR, is provided in the [4].

```
IMPORT   omorig_group_t    *omorig_get_first_group (
   IN void             *set);
```
Returns a pointer to the first group in the DB.

```
IMPORT   omorig_group_t    *omorig_get_next_group (
   IN void             *set,
   IN void             *elem );
```
Returns a pointer to the next after the specified one group in the DB.

```
IMPORT   vr_descriptor_t      *omdb_create_vr (
   IN uint32_t         vpn_id,
   IN ipaddr_t         *vr_id   );
```
Creates VR descriptor for specified vpn id and vr id and fills in with default values.

```
IMPORT   omorig_group_t    *omdb_create_group (
   IN uint32_t         vpn_id,
   IN ipaddr_t         *vr_id,
   IN int              class_selector_flag  );
```
Creates Group descriptor for specified vpn id and vr id and fills in with default values. Class selector flag defines set of object classes, which are mandatory to be created for group to be created.

```
IMPORT   int omdb_destroy_vr (
   IN uint32_t      vpn_id,
   IN ipaddr_t      *vr_id    );
```
Destroys VR descriptor for specified vpn id and vr id.

```
IMPORT   int   omdb_delete_group (
   IN  void    *arg1,
   IN  void    *arg2 );
```
Destroys Group descriptor after all the object destoyed.

```
IMPORT omorig_group_t      *omorig_lookup_group_by_id (
   IN object_group_id_t group_id );
```
Looks up group by the specified group ID.

```
IMPORT   int   omorig_add_obj_id  (
   IN object_id_t       *obj_id  );
```
Adds object ID to the OM Global Database.

```
IMPORT    int    omorig_remove_obj_id (
   IN void              *flag,
   IN void              *obj_id  );
```
Remove object ID from the OM Global Database.

```
IMPORT    oid_link_t   *omorig_lookup_oid (
   IN object_id_t        *id   );
```
Finds OID link in the OM Global Database by the specified object ID.

```
IMPORT int om_create_vpn (
   IN uint32_t          vpn_id);
```
Creates VPN descriptor and fills in with default values.

0095384.070301